NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

THESIS

MIGRATING DEPARTMENT OF DEFENSE (DOD) WEB SERVICE BASED APPLICATIONS TO MOBILE COMPUTING PLATFORMS

by

Paul Andrew Lafranchise

March 2012

Thesis Advisor:	Thomas Otani
Second Reader:	Warren Yu

Approved for public release; distribution is unlimited

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704–0188

1. REPORT DATE *(DD–MM–YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From — To)*
23–3–2012	Master's Thesis	2102-06-01—2104-10-31

4. TITLE AND SUBTITLE

Migrating Department of Defense (DoD) Web Service Based Applications to Mobile Computing Platforms

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)

Paul Andrew Lafranchise

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Naval Postgraduate School
Monterey, CA 93943

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Department of the Navy

10. SPONSOR/MONITOR'S ACRONYM(S)

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION / AVAILABILITY STATEMENT

Approved for public release; distribution is unlimited

13. SUPPLEMENTARY NOTES

The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.IRB Protocol Number: n/a

14. ABSTRACT

This thesis explores techniques for extending Department of Defense (DoD) Information Technology (IT) capability from web-based desktop clients to mobile platforms. Specifically, we examine how existing data services can be consumed by native and web-based mobile clients without modification to the services. We consider the data access layer, the User Interface (UI) design, and the Total Cost of Ownership (TCO) as areas to compare and contrast each implementation. We develop a web-based application and implement comparable capability on native and web-based mobile clients. We determined that native applications and mobile aware web applications are capable of consuming an existing web service without modifying the service. In general, we find no clear advantage between our mobile implementations when consuming existing web services and maintaining a consistent UI. We found that, while developing a data access module, it is difficult to share code between an existing web application and a native mobile application. We find that in some cases, a mobile aware web application excels at rapidly deploying on multiple devices and because it maintains a single code baseline lowering the TCO.

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			19b. TELEPHONE NUMBER *(include area code)*
Unclassified	Unclassified	Unclassified	UU	51	

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release; distribution is unlimited

MIGRATING DEPARTMENT OF DEFENSE (DOD) WEB SERVICE BASED APPLICATIONS TO MOBILE COMPUTING PLATFORMS

Paul Andrew Lafranchise
Civilian, BAE Systems
B.S., California State University, San Marcos, 2011

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
March 2012

Author: Paul Andrew Lafranchise

Approved by: Thomas Otani
Thesis Advisor

Warren Yu
Second Reader

Peter J. Denning
Chair, Department of Computer Science

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

This thesis explores techniques for extending Department of Defense (DoD) Information Technology (IT) capability from web-based desktop clients to mobile platforms. Specifically, we examine how existing data services can be consumed by native and web-based mobile clients without modification to the services. We consider the data access layer, the User Interface (UI) design, and the Total Cost of Ownership (TCO) as areas to compare and contrast each implementation. We develop a web-based application and implement comparable capability on native and web-based mobile clients. We determined that native applications and mobile aware web applications are capable of consuming an existing web service without modifying the service. In general, we find no clear advantage between our mobile implementations when consuming existing web services and maintaining a consistent UI. We found that, while developing a data access module, it is difficult to share code between an existing web application and a native mobile application. We find that in some cases, a mobile aware web application excels at rapidly deploying on multiple devices and because it maintains a single code baseline lowering the TCO.

THIS PAGE INTENTIONALLY LEFT BLANK

Table of Contents

List of Figures

THIS PAGE INTENTIONALLY LEFT BLANK

List of Tables

THIS PAGE INTENTIONALLY LEFT BLANK

List of Acronyms and Abbreviations

API Application Programming Interface
CSS Cascading Style Sheets
CLIMO Climatology Data
COLD-T Combat Operator's Load-out Decision Tool
C2 Command and Contro
CORE Common Operational Research Environmentl
CM Content Management
DOD Department of Defense
DWML Digital Weather Markup Language
FNMOC Fleet Numerical Meteorology and Oceanography Center
GAE-J Google App Engine for Java
GUI Graphical User Interface
GWT Google Web Toolkit
HTML Hyper Text Markup Language
iOS iPhone Operating System
IT Information Technology
JAX-WS RI Java API for XML Web Services Reference Implementation
JS JavaScript
JSNI JavaScript Native Interface
METOC Meteorological and Oceanographic
MAA Mobile Application Archetype
MVC Model View Controller
NDFD National Digital Forecast Database
NOAA National Oceanic and Atmospheric Administration's
NSSC Natick Soldier Systems Center
NFC Near Field Communications
NOGAPS Navy Operational Global Atmospheric Prediction System
NPS Naval Postgraduate School
NSW Naval Special Warfare
OGC Open Geospatial Consortium
OS Operating System
PCU Protective Combat Uniform
PoR Program of Record
REST REpresentational State Transfer
RE Runtime Environment
SAX Simple API for XML
SDK Software Development Kit
SOA Service Oriented Architecture
SOAP Simple Object Access Protocol

SOF Special Operations Forces
TCO Total Cost of Ownership
UI User Interface
USG United States Government
W3C World Wide Web Consortium
WSDL Web Service Definition Language
XML Extensible Markup Language

CHAPTER 1:
Introduction

This thesis explores techniques for extending Department of Defense (DoD) Information Technology (IT) capability from web-based desktop clients to mobile platforms. Specifically, we examine how existing data services can be consumed by native and web-based mobile clients without modification to the services. We accomplish this by developing a web-based application and implementing it on native and web-based mobile clients. We consider the data access layer, the User Interface (UI) design, and the Total Cost of Ownership (TCO) as areas to compare and contrast each implementation.

The last five years of DoD IT development has been dominated by the goal of creating a Service Oriented Architecture (SOA) to decouple applications from their data and expose the data with visible, accessible, understandable and trusted data services. The DoD Information Management Strategic Plan 2008-2009 states: *The effort to accelerate DoD's net-centric information sharing transformation is dependent upon adoption of technologies that support two overarching concepts: Service-Oriented Architecture (SOA) and Web 2.0.* [1]. As a result of this effort, there exist a large number of DoD IT data services that can be combined to quickly create new applications.

To date, these SOA services have been processed or consumed by regular desktop applications, often web-based. As mobile computing devices become more ubiquitous in our everyday life there is an increasing expectation that DoD IT application capability make the transition from the traditional desktop applications to the diverse emerging mobile platforms. The combination of readily available data services and increasingly powerful mobile platforms make this a very exciting time to be a software developer as the DoD is poised on the brink of an explosion of mobile computing application development.

There are two major approaches to developing applications for mobile platforms that are most promising for exposing SOA services to much wider audience. These approaches are developing native mobile applications and developing web-based mobile applications. Native applications can be developed with Software Development Kits (SDK) available from each of the major mobile device manufacturers and which allow developers to design and deploy application binaries that run in the native machine format of the device. Web-based mobile applications

offload the device independence to the mobile browser and allow developers to create a single application that identifies mobile platforms and adapts it's content appropriately.

To effectively support native mobile applications, desktop web-based applications, and mobile web-based applications consuming services from the same source, there are a number of technical issues we must address. The standard mobile application considerations of computation power, occasional connectivity, and limited bandwidth are well documented and discussed in Chapter 2. In Chapters 3 and 4, we consider the client data access layer, the application UI design, and TCO when extending an existing service based application to multiple, differing clients. We determine if the native applications we develop are capable of consuming an existing web service (without modifying the service), and then we explore the possibility of sharing the client-side data access code between application implementations. As we develop each mobile client, we attempt to create a UI which has a look, feel, and work-flow that matches the original web application. Finally, we discuss TCO. Ideally, we should not be required to develop three completely different versions of the application, one version for each platform. Is there a right way to maximize the code reuse to reduce the development time and the maintenance cost? What can be shared and what cannot are not obvious as the tools, languages, etc. are quite different for the three platforms.

1.1 Case Study

As a case study we implement a decision aide application for the Naval Special Warfare (NSW) Cold Weather Detachment, Kodiak, AK. The application, named the Combat Operator's Load-out Decision Tool (COLD-T), consumes data from Fleet Numerical Meteorology and Oceanography Center (FNMOC) and must be accessible from Android, iPhone Operating System (iOS) devices and standard web browsers. We take application requirements directly from NSW personnel and from a FNMOC White Paper authored by LT Geoffrey D. Goldstein USN [2]. The COLD-T application serves as a tactical level mission planning tool. The application presents the user with a map interface on which they can plan a mission route. The user is presented with information on their route in real-time as they plot it including: weather conditions, geolocation, elevation, and a 2D route profile. When the user completes their route, the application presents the user with a gear list based on the route profile and the predicted weather conditions along the route. The application consumes three different services:

- Weather Data: A SOAP web service that returns predicted weather parameters for a given latitude, longitude, elevation and date-time based on FNMOC's most accurate weather

model for the region.

- Climatology Data (CLIMO): A SOAP web service that returns predicted weather parameters for a given latitude, longitude, elevation and date-time based on FNMOC's historical average weather model for the region. CLIMO data is used to make gear recommendations when the requested date-time is outside of FNMOC's prediction window.
- Elevation Data: A SOAP web wervice that returns the elevation in meters for a given latitude and longitude. Elevation data is used to create a 2D elevation graph of the planned route.

We use the COLD-T application as the case study because it illustrates the challenges faced by DoD IT Programs of Record (PoR) on a small scale. Namely, FNMOC has data that has been made available through services and the COLD-T application is required to have a client or clients that access the FNMOC data on traditional and mobile platforms. Additionally, FNMOC is co-located with NPS, and partnership between NPS and FNMOC benefits both organizations. NPS can access real world data and requirements, while FNMOC can leverage NPS research to solve some of its hard technical problems.

1.2 Approach

We investigate different implementation approaches for the emerging breed of DOD applications that are required to span multiple mobile operating systems and traditional web applications. We break this problem down into two general areas, and for each area, we study different approaches and propose the best approach. The areas are as follows:

1. Data layer: How to best architect a data access package. Can today's most popular mobile operating systems, Android and iOS, natively consume Simple Object Access Protocol (SOAP) based Meteorological and Oceanographic (METOC) web services or will architectural concessions need to be made? We develop data clients, for FNMOC's SOAP service, on web based, native, and web based mobile applications. We confirm that accessing the service is possible on each platform and then explore how we can create data clients that can be shared among some or all platforms.
2. User interface: Which tenets of proper UI design can be applied in order to create a consistent and recognizable user experience across mobile and browser based clients? We develop the user interface for the COLD-T web application first and then duplicate the web application's functionality on a native device and as a mobile aware web application. We explore methods to maintain a recognizable user experience across each implementa-

tion. We compare and contrast the advantages of developing a user interface with a native device API and a cross-platform development technology.

As discussed in section 1.0, we study these two areas because they are not just specific to mobile application development but also to our particular problem—extending a data service driven application to multiple mobile clients while maintaining the application's work-flow. Finally, we observe that developing a native application capability for every mobile device environment is often cost prohibitive, especially from a DOD acquisition perspective. We identify application requirements that are appropriate for emerging cross-platform development solutions and detail how our cross platform version of the COLD-T application (a mobile aware web-application) excels in both of our areas of inquiry.

CHAPTER 2:
Background

Research into mobile application development can be broadly categorized into three areas: how to create resource-efficient mobile applications, how to create cross platform mobile applications, and how to expose enterprise data and capability on mobile platforms. We explore each of these areas and then show how extending existing DoD web applications to mobile platforms necessitates combining and building on each area.

2.1 Resource-Efficient Applications

There is an abundant amount of research into developing mobile applications that address the common constraints of the mobile computing environment—limited bandwidth, limited computing power, and limited connectivity. Mobile devices are generally characterized by the following features as discussed in [3].

- (Pros) Flexible wireless network access (Wi-Fi, CDMA, GSM, 3G, etc.)
- (Pros) Capability to sense the device's context
- (Pros and Cons) High tendency of device mobility
- (Cons) Potential inability to access a network
- (Cons) Limited resources and computing power

The advent of several open Software Development Kits (SDK)s for popular mobile operating systems, iPhone and Android for example, has led to an explosion of mobile applications. One of the main draws of these SDKs is that they allow developers to create *native* applications which make use of the benefits of mobile platforms (access to native APIs) while minimizing the cons (built-in power and resource management). Consequently it is common for an organization to maintain a separate development branch for each target mobile (and/or web) platform on which they wish to deploy [4].

2.2 Cross Platform using Web Technology

No single mobile Operating System (OS) dominates the mobile market nor can we expect one to in the foreseeable future. Many mobile OSs are a challenge for software developers and, by extension, the DoD as many of it's Programs of Record (PoR) stand on the brink of moving into

mobile application development. Challenges to mobile application developers due to the ever increasing number of mobile OSs include the following as discussed in [**?**] [5]:

1. Software developers must expend time learning each OS's SDK and APIs as new smart phone platforms are released.
2. With the emergence of each new smart phone OS application, functionality must be redeveloped.
3. The Total Cost of Ownership (TCO) of the application is high. Updates to the application must be made to all versions of the application; it is difficult to maintain consistency across all versions of the application.

Because these are such serious problems, being able to write once, compile to, and run on many mobile systems is a coveted, and increasingly necessary capability. One of the most popular approaches to solving this problem is to create a framework in which applications can be developed in a common language and then compiled into native code or assembled in a cross platform Runtime Environment (RE) [6] describes just such a framework with 4 layers: Application Layer, JavaScript (JS) Engine Layer, Component Layer, and OS Layer. The *Application Layer* is comprised of web based languages (HTML, XML, JS, etc.) and used to develop the application functionality. The *JS Engine Layer* provides a uniform interface for the *Application Layer* to invoke the component layer. The *Component Layer* is a set of pre-compiled libraries that expose device specific capability. *The Component Layer* is swapped out based on the native OS. The *OS Layer* represents the device specific APIs exposed by each native OS.

Rhodes (Rhomobile) is a commercially available implementation of this type of framework. Using the Ruby programming language, program developers can write applications that compile to *native* code for most major mobile OSs (iPhone, Android, RIM, Windows Mobile and Windows Phone 7). One of the best features of the Rhodes API, besides being cross platform, is the ability to access device capabilities such as GPS, PIM contacts and calendar, camera, native mapping, push, barcode, signature capture, Bluetooth and Near Field Communications (NFC).

2.3 Enterprise Data for Mobile

The last 5 years of DoD software development has seen a dramatic push for Service Oriented Architectures (SOA)s as tools to expose data from legacy information systems. As enterprise services have become increasingly common, research has begun to explore how to expose these services and their data on mobile platforms in an effective manner. This research has shown

that it is necessary to minimize data transferred to and stored on the device, take steps to load data pro-actively, and allow applications to fully function in a disconnected mode [5]. The basic requirements for such a mobile SOA solution should include the following: 1) timely, robust and easy access data source, 2) transparency between connected, occasionally-connected, and disconnected modes, 3) a loose-coupling system designed to combine services on demand, 4) lightweight application composition and development and, 5) low total cost of ownership [5].

Mobile SOA solutions can be broadly defined as *online* and *offline*. An *online architecture* assumes that the connection between mobile devices and backend systems is available most of the time. It uses a typical client/server model to build online applications. A mobile online application is normally a thin client that only processes the presentation logic and accesses the business logic on the server side using remote invocations. The major function of the online architecture is to support these remote invocations in the mobile environment [7]. While *offline architecture* assumes that the connection between mobile devices and backend systems is unavailable most of the time. It is usually data centric. A mobile offline application is normally a fat client that processes both the presentation and business logic locally on mobile devices with local business data that is downloaded from backend systems. The major function of the offline architecture is to periodically synchronize data between the client and backend systems [7].

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 3:
Description of the Work

We believe that current DoD service driven web applications must be executable not only on traditional computing systems but on the many, increasingly capable, and common mobile computing platforms. We explore different methods of extending application capability to mobile platforms by implementing the COLD-T application as a traditional web application and then extending it to a native mobile application and a mobile aware web application. Each of these applications retrieves its data from a single enterprise web service.

- We define a traditional web application as an application which is developed in a browser-supported language, displayed with a browser-rendered markup language, served by an application server, and executed in a common web browser.
- We define a native application as an application designed to run in a native mobile environment (machine language and OS) able to make full use of the OS's device APIs.
- We define a mobile aware web application as an application which is developed and served like a traditional web application but is capable of identifying the client's browser (and device) by the user-agent value and adapting its content and behavior accordingly (including accessing native device APIs).

3.1 Description of the Application

The COLD-T application is a decision aid for Special Operations Forces (SOF). The SOF user plots the route of a planed patrol by dropping waypoints on a 2 dimensional (2D) map in the COLD-T UI. The SOF user provides a date-time and posture for each waypoint on the planed route. As a waypoint is plotted, the application asynchronously queries an enterprise web service for the weather conditions at the waypoint during the supplied date-time. Once the predicted weather data is returned, the COLD-T application makes a decision about which uniform layers are appropriate for the mission given the route, posture, and weather. The algorithm used to make the prediction is based on the specifications for the Protective Combat Uniform (PCU) developed by the U.S. Army's Natick Soldier Systems Center (NSSC), Natick, Massachusetts. The weather conditions required to make the prediction are minimum temperature, maximum temperature, and percent precipitation.

3.2 Traditional Web Application

We chose to implement the web application with Google Web Toolkit (GWT) and deploy it on Google App Engine for Java (GAE-J). GWT and GAE are not required to implement the COLD-T application, any web framework (PHP, Struts, JSF, Spring MVC, etc.) could be used. We chose GWT because of familiarity with the Java programming language and because GWT produces less error prone and (most importantly) cross browser compatible JavaScript. Additionally, GAE allowed us to easily and quickly deploy the COLD-T web application to a location accessible by our test user-base in Kodiak, AK. UI elements and data requests and responses are serialized between the client (browser) and server. Requests of the weather service are made server-side and then serialized and returned to the client (browser). Figure 3.1 shows a class diagram of the classes and interfaces involved in the weather request.

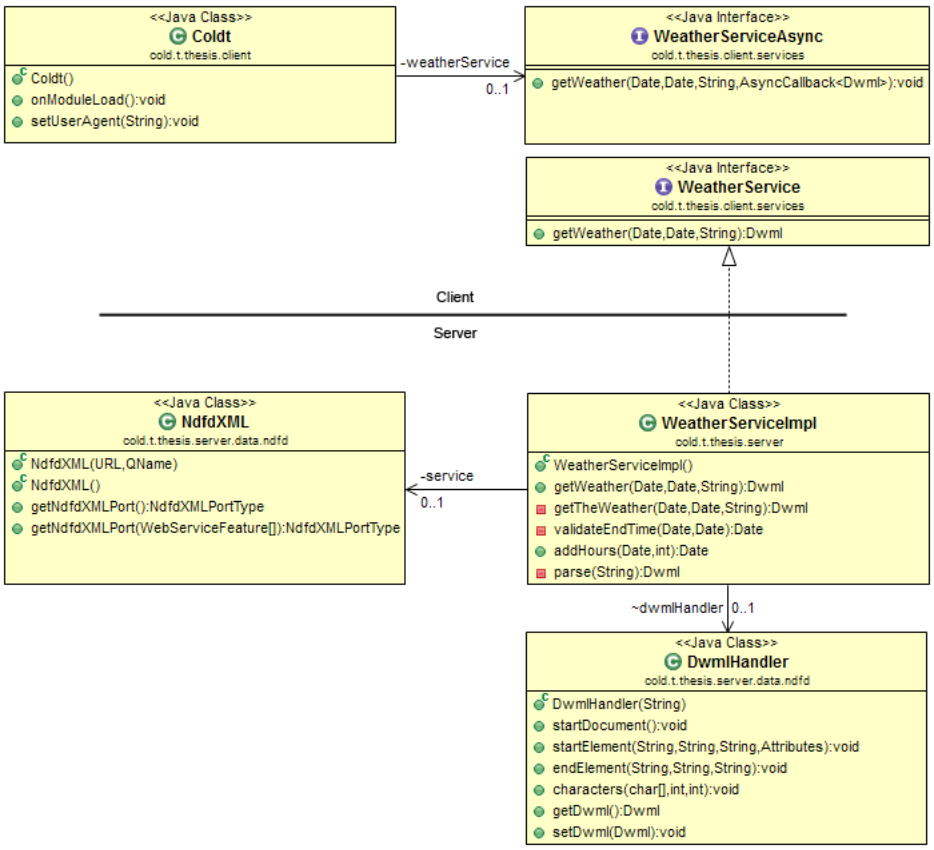

Figure 3.1: *The class diagram for classes involved in a weather request in the COLD-T web application.*

These classes provide the necessary functions to make a weather request of the NOAA SOAP web service from the COLD-T web application. Table 3.1 gives a brief description of each of the classes.

Table 3.1: *The core classes for the COLD-T web application.*

Class	Description
Coldt	This class is the application's entry point it implements the *EntryPoint* interface by defining an *onModualLoad* function. The *onModualLoad* function is called as soon as both the module and the outer document is ready. The *Coldt* class is a client-side class and is compiled to Javascript when the application is built.
WeatherServiceAsync	This interface defines the *ServiceDefTarget*, an asynchronous interface to the weather service that can be called by the client-side Javascript code.
WeatherService	This interface extends RemoteService and defines the *getWeather()* service RPC method.
WeatherServiceImpl	This class extends RemoteServiceServlet and implements the *WeatherService* interface. It is the server-side service that receives RPC requests from the client-side code.
NdfdXML	This is a class generated by JAX-WS RI from the NOAA NDFD weather service WSDL. It exposes the web service's functions and marshals (and unmarshals) request and responses to (and from) the web service functions.
DWMLHandler	This class implements a SAX XML parser. It parses the web service response from an XML string into a POJO which is returned by the RPC service.

Figure 3.2 depicts a sequence diagram for the COLD-T web application from page load through a weather request. The *Coldt* class serves as the application entry point, its *onModuleLoad* function sets up the application's UI and click handlers. The user plots a route on the map by centering the map over the desired location and selecting the *Add Waypoint* button. A form associated with the *Add Waypoint* button allows the user to enter the data associated with the waypoint: name, date and time, and posture. The click handler receives the click event and creates a new waypoint, adds it to the waypoint table, and sends an asynchronous weather request to the server-side weather service. The weather service marshals and sends a SOAP request to the weather web service and unmarshals the response. The response body is passed to our Simple API for XML (SAX) handler that parses the XML into Java objects. Finally, the weather service serializes and sends a response to the client-side click handler. If the server-side

actions completed successfully, the response is a serialized Java object containing the weather parameters. The click handler updates the waypoint table with the weather data. If the server-side actions fail, the response contains a *Throwable* object with the error message and the click handler displays an error dialog.

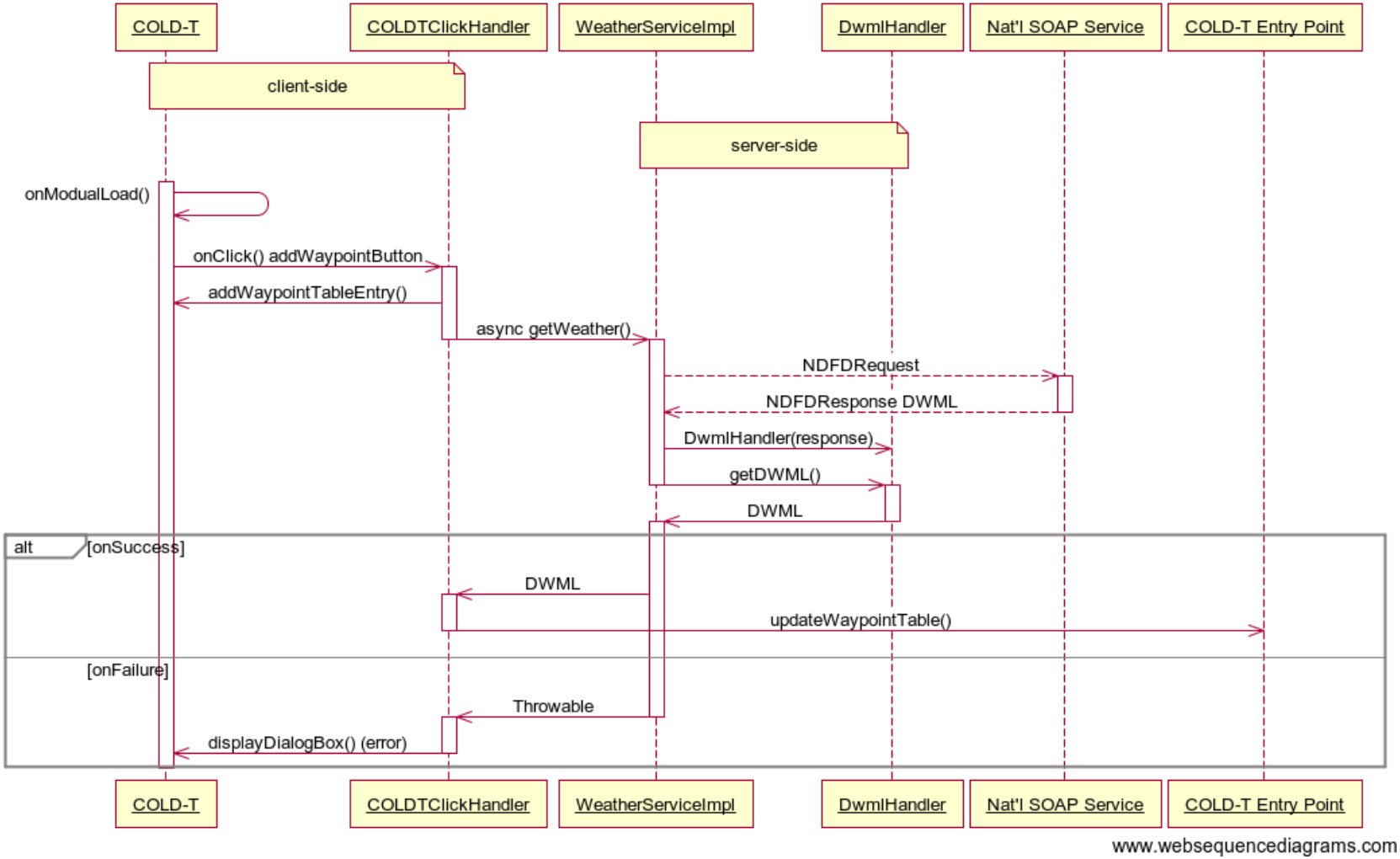

Figure 3.2: *The sequence diagram for a weather request from the COLD-T web application to the NOAA web service and back.*

3.3 Native Mobile Application

We chose Android as the mobile application development framework for the COLD-T native application. This selection was made primary because of the author's familiarity with the Java programming language and because of the high probability of successfully reusing Java components between the web and native applications. Additionally, the Android OS has been adopted by the Defense Advanced Research Projects Agency (DARPA) and the National Security Agency (NSA)[1] [8]. Unlike the traditional web application, which serves the executable client code and markup to the client browser, there is no client/server separation in the COLD-T Android application. The application runs entirely on the mobile device as an application in the Android OS. Calls to the weather service are made directly from a worker thread in the COLD-T application on the accessing device. Figure 3.3 is the class diagram of the COLD-T native mobile application.

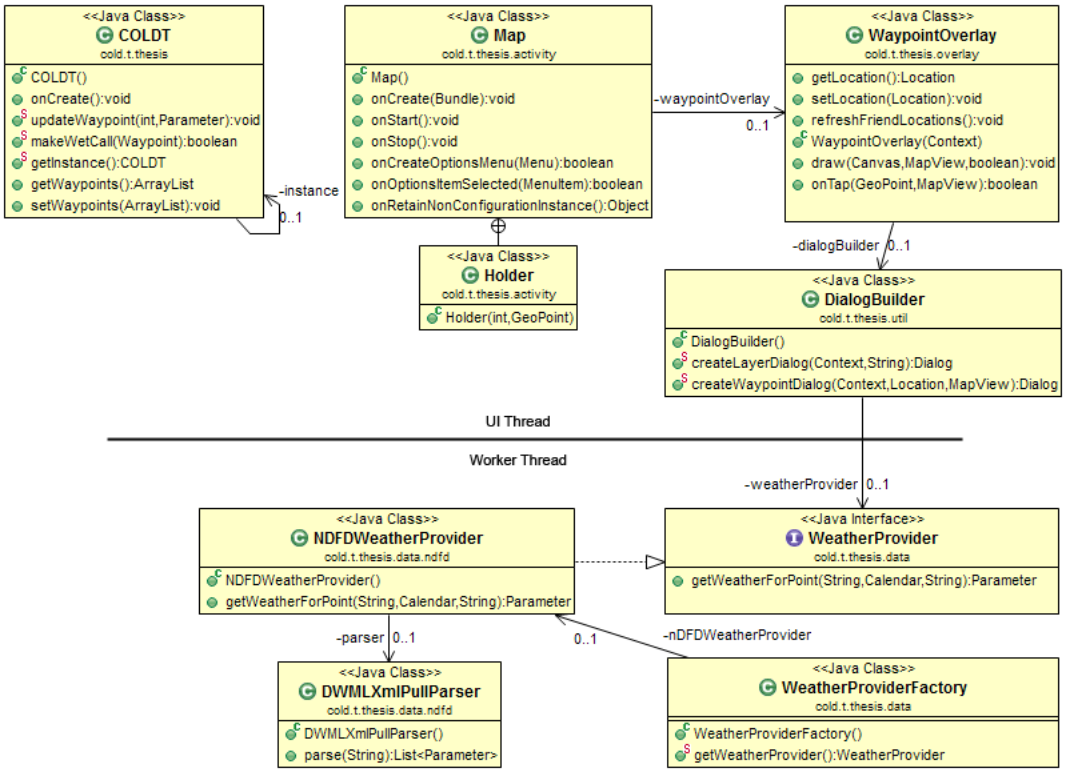

Figure 3.3: *A class diagram of a COLD-T Android application weather request.*

[1]DARPA modifies the Android kernel to meet NSA standards for processing classified data. Once certified by the NSA, the modified OS can be used by any government agency.

These classes provide the necessary functions to make a weather request of the NOAA SOAP web service from the COLD-T native mobile application. Table 3.2 gives a brief description of each of the classes.

Table 3.2: *The core classes for the COLD-T native mobile application.*

Class	Description
COLDT	This class is a base class for maintaining global application state. It maintains the list of waypoints and associated weather data so that application state can be preserved on orientation changes and between the map and list views.
Map	This class extends MapActivity which provides a map view and geo-spacial operations. It contains a Holder class to maintain map state (position, zoom level) across orientation changes.
WaypointOverlay	This class handles map clicks, initiates the add waypoint dialog, and provides a surface to paint the route polyline.
DialogBuilder	This class constructs the add waypoint dialog, takes the user's input, and initiates the GetWeatherTask thread to retrieve the weather for a new waypoint.
WeatherProvider	This interface defines an abstracted weather provider interface that can be implemented by multiple data providers.
WeatherProviderFactory	This factory class identifies and constructs the appropriate weather provider class.
NDFDWeatherProvider	This class implements the weather provider interface. It uses the kSOAP library to make requests to, and receive responses from the NOAA DWML web service.
DWMLXMLPullParser	This class implements a *org.xmlpull.v1.XmlPullParser* to parse the DWML response from an XML string to a POJO.

Figure 3.4 depicts a sequence diagram for the native application from startup through a weather request. The application is initiated once the user selects the application icon. The COLD-T application starts a UI thread and displays the user interface. The user plots a route on the map by selecting waypoints with a touch on the map. Each touch event brings up the "Add Waypoint" dialog box. From the dialog box, a user enters a name for the point, a date and time, and their planned posture. Once the user completes the "Add Waypoint" dialog, they are returned to the map view while the application spawns a worker thread to retrieve the weather data for the new waypoint in the background. We use the Factory Pattern to choose the weather

provider.[2] Once the worker thread has a weather provider, it invokes the provider's *getWeatherForPoint()* function. The weather provider marshals the SOAP request, sends the request, and unmarshals the response. The response is passed to the XML parser which parses out the weather data for the point and returns a list of weather parameter objects. The weather provider returns the most current weather parameter to the worker thread which updates the new waypoint (on the UI thread) with the weather data. Alternatively, if there is a problem with the weather service request, the weather provider throws an *IllegalArgumentException*, and the worker thread updates the UI thread with an error message in a *toast* display.

[2]Currently, the only available weather provider is the NOAA's National Digital Weather Forecast. FNMOC is developing a weather service for COLD-T. Once complete, we will be able select between both weather providers.

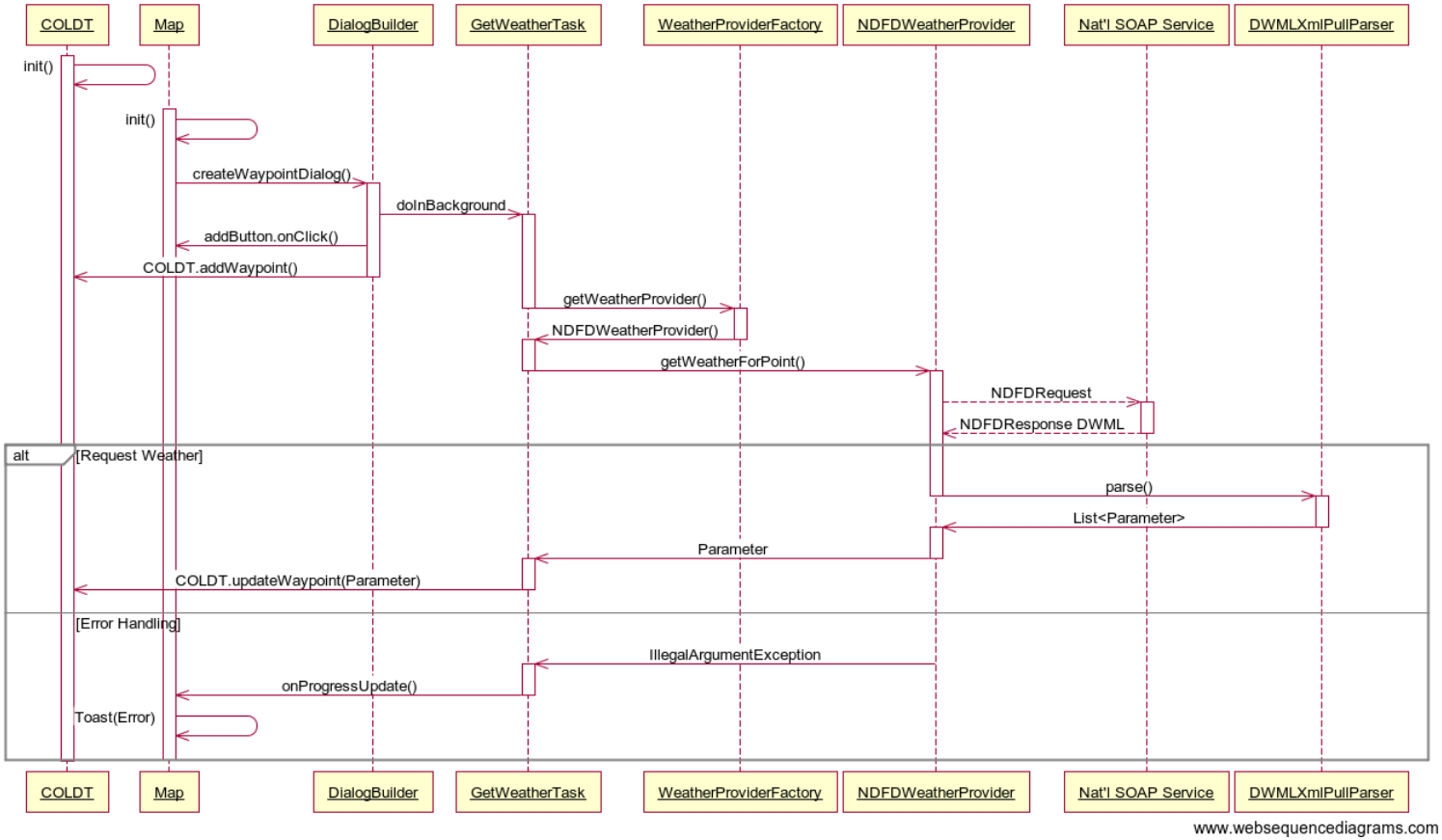

Figure 3.4: *The sequence diagram for a weather request in the COLD-T native application client to the NOAA web service and back.*

3.4 Mobile Aware Web Application

The mobile aware web based implementation of COLD-T is an extension of the web based implementation that is designed to perform on a mobile device. The data retrieval sequence diagram is identical to that of the web application—all the changes are made on the client-side code. We update the map, add geo-location awareness, and add conditional CSS. We migrate the map API from Google Maps JavaScript API v2 to v3 in order to take advantage of its mobile friendly features. This newer version of the API is capable of handling mobile specific input like pinch-to-zoom which allows mobile users to interact with the map just like they would with a native map application. To highlight the new map, we make use of the World Wide Web Consortium (W3C) Geolocation API to identify the device's location and then center the map on the device. Finally, we modify the entry point of the web application to identify the client's browser and then tailor the application's appearance accordingly using conditional CSS. The User-Agent HTTP request-header is designed for this purpose; it contains multiple product tokens with information identifying the user agent originating the request. The following code snippet shows the JavaScript Native Interface (JSNI) function that retrieves the user-agent value.

```
public static native String getUserAgentString() /*-{
  return $wnd.navigator.appName + ' : ' +
    $wnd.navigator.userAgent.toLowerCase();
}-*/;
```

Here is an example user-agent value retrieved by this function for an iPad running mobile Safari.

```
Netscape : mozilla/5.0 (ipad; cpu os 5_0_1 like mac os x)
applewebkit/534/46 (khtml, like gecko) version/5.1 mobile/9a405
safari/7534.48.3
```

Once a mobile user-agent has been identified, the conditional CSS styles the page in a mobile friendly manner; white space is removed and the map and input panels are sized to fit the entire page. Here is the conditional CSS statement for loading mobile CSS.

```
/** Execution time css - used to load mobile css */
@if (cold.t.thesis.client.EntryPoint.isMobile()) {
    ... mobile css
}
```

Using conditional CSS to build the COLD-T mobile aware web application's UI for traditional and mobile applications leaves the client-side application logic unchanged and allows us to keep our browser-specific css in the same file, reducing complexity and page load time. Figure 3.5 shows the different client UIs generated by the conditional CSS. The traditional web application view is on the left while the mobile view is on the right.

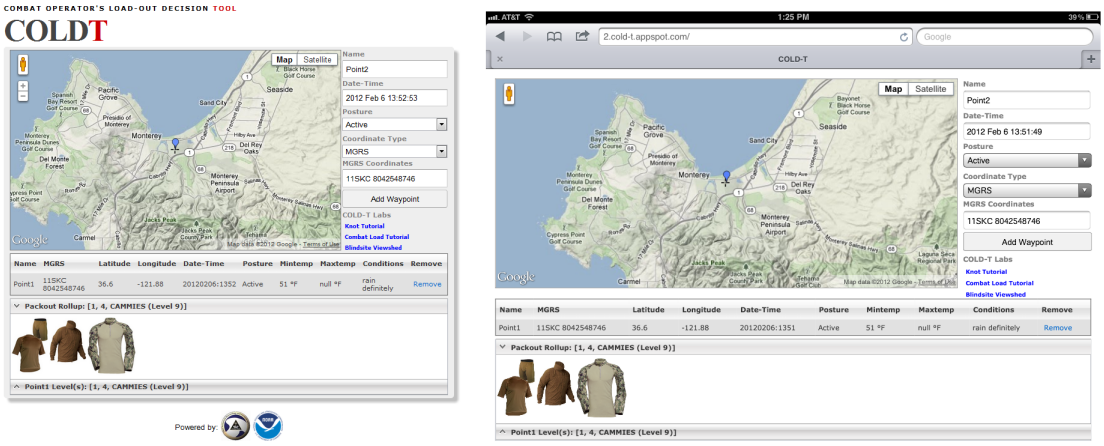

Figure 3.5: *The different UIs created by the COLD-T mobile aware web application's conditional CSS.*

3.5 Data Access Module

The Fleet Numerical Meteorology and Oceanography Center (FNMOC) Monterey, California generates a global numerical weather model called the Navy Operational Global Atmospheric Prediction System (NOGAPS). This weather model data will be available via Open Geospatial Consortium (OGC) compliant web services in September of 2012. In order to meet thesis deadlines, we use the National Oceanic and Atmospheric Administration's (NOAA) National Digital Forecast Database (NDFD) SOAP web service as our weather source for each COLD-T application implementation, Figure 3.6.

Because both Android and GWT/GAE place restrictions on the Java dependencies that can be used in their respective environments, we are not able to share a single data client between the native and web applications. The NDFD's Web Service Definition Language (WSDL) file defines its request and response elements as Digital Weather Markup Language (DWML) elements but defines the response type as *type=xsd:string*. That is, the NDFD response contains well formed XML but is typed as a string. Because of this, we cannot use automated techniques to parse the response XML to objects, we must write custom XML parsers for each data client.

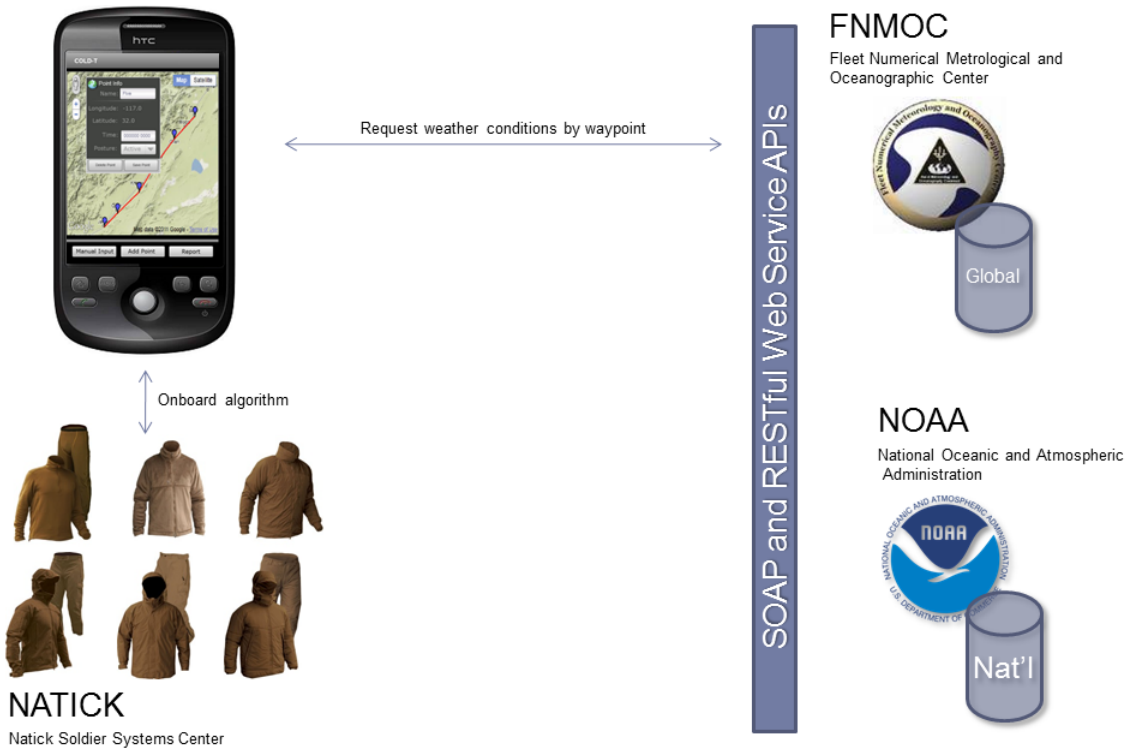

Figure 3.6: *High level COLD-T data access architecture.*

3.5.1 Web Data Layer

The COLD-T web application data client exposes the public method *getWeather* that takes as arguments: a start time, an end time, and a string representation of the route coordinates.

```
public Dwml getWeather(Date startTime,
    Date endTime, String latLonString)
```

The marshaling and un-marshaling of the request and response is performed by classes generated from the NDFD WSDL using the Java API for XML Web Services Reference Implementation (JAX-WS RI). A custom *javax.xml.parsers.SAXParser* parser is used to parse the DWML response string into Plain Old Java Objects (POJOs). Serialized POJOs are then returned to client (browser).

3.5.2 Native Data Layer

The COLD-T native application data client exposes the public method *getWeatherForPoint* that takes as arguments a string representation of the waypoint coordinates and a date-time. We changed the request mechanism in the native application to request data for each point individ-

ually. This was done to gain more accurate weather data about the waypoint and date-time.

```
public Parameter getWeatherForPoint(String latlon,
    Calendar dateTime, String options)
```

Google does not provide native Android support for SOAP; they prefer to use RESTful web services. Third party libraries exist however, and we use the kSOAP library to manually build the NDFD SOAP request and perform the send and receive mechanics. Android does provide an XML parser, *org.xmlpull.v1.XmlPullParser*. We use this interface to parse the response string and build the response objects.

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 4:
Analysis of the results

While designing and implementing the COLD-T application as a web application, a native mobile application, and as a user agent aware mobile web application, we focus on answering the following questions:

- Primary research question: Is it possible to exploit existing web service capability to create applications for multiple mobile operating systems and browser based clients?
  - Subsidiary research question 1: What design pattern is capable of providing data to create a data layer that spans multiple mobile operating systems and browser based clients?
  - Subsidiary research question 2: What techniques can be employed to build user interface that is consistent and recognizable across platform technologies?

4.1 Primary Research Question

We find that it is possible to exploit existing web service capability when creating applications for multiple mobile operating systems and browser clients. As detailed in Chapter 3, we demonstrate two different methods of extending the COLD-T application to mobile devices: 1) an Android based native application that duplicates the functionality of the COLD-T web application with native OS APIs (Section 3.3) and 2) a mobile aware web application that adapts its content for mobile devices (Section 3.4). Both of these approaches enable the application to query for and retrieve data from the existing NOAA DWML web service without modification to the service (Section 3.5).

Each approach has its benefits and costs. Building native implementations of existing application capability provides unlimited access to the device's capabilities which can add rich new features to the application. For example, direct access to native devices like the camera is trivial when developing a native application. In contrast, the W3C device API is, as of 18 September 2010, only in working draft. The drawback to native applications is that a new version must be developed for each target platform. Each new native implementation increases Content Management (CM) costs as new software baselines are created for each supported native device. Adapting an existing web application to identify mobile devices in order to tailor the application's behavior and content decreases CM costs and moves the burden of device independence

onto the browser. As previously stated, mobile browser-to-device APIs are still developing and this approach often comes at the cost of limited access to some of the native device's capability.

We find no clear advantage between these two approaches in general. Advantages, however, can be found when considering migrating a specific capability to one or several mobile platforms. Weighing the planned application capability requirements with the target device capability, the target network characteristics, and end user's preferences can help identify an implementation approach. For the COLD-T application, because we can take advantage of the mobile browser's ability to provide geo-location and native storage, we find that a mobile aware web application is the most attractive option for quickly and cheaply extending the COLD-T capability to multiple mobile platforms.

4.2 Subsidiary Research Question 1

As we show in section 4.1, a single server-side web service can be used by different types of clients. In an attempt to share code across application implementations, we explore the creation of a common data access module for the NOAA weather SOAP web service that can be used by both our web and native clients.

4.2.1 Native Mobile Implementation

We chose to develop our native application with Android, because Android has a Java based development environment and can potentially share code with our Java based web application. We try three different Java based SOAP utilities to create a data client that can be shared between the Java based web application and the Android application: XMLBeans, JAX-WS, and kSOAP. XMLBeans and JAX-WS are both WSDL to Java technologies that will generate Java classes from the SOAP Web Service Description Language (WSDL) document. kSOAP is a SOAP web service client library for constrained Java environments. Initial attempts to use an XMLBeans based client showed that the XMLBeans *.class* files will not cross compile to Dalvik on the Android platform. We were able to use the JAX-WS based client on the web based implementation, but JAX-WS generated classes will not work on the Android platform. We use a kSOAP based client on the native application. Though it may be possible to share code between a Java based web application and a Java based native mobile application, in practice, the constraints imposed by each environment make it difficult to share WSDL to Java based web

service clients[3]. We find that, where possible, the benefits realized by sharing a data client between a Java based web application and an Android application would be minimized with each additional implementation (iOS, Windows 7, etc.).

4.2.2 User Agent Aware Web Based Implementation

As described in section 3.4, the COLD-T user agent aware web application recognizes when the client browser is running on a mobile device and adapts the UI and behavior of the application for a mobile environment. As shown in figure Figure 3.1, the data client runs entirely on the server-side and is evoked through asynchronous HTTP requests from the client-side code. This abstracts the data client from the client-side code allowing a single data client to be shared across each recognized user-agent (mobile browser). In fact, all the server side code is "shared" across each client.

4.3 Subsidiary Research Question 2

In order to explore how we can maintain program continuity across COLD-T implementations, we develop the COLD-T web application as a baseline and then attempt to maintain a consistent and apparent user interface for native mobile and mobile aware web versions of the application.

4.3.1 Web-based Implementation

The COLD-T web application is a "single page" web application; once the application loads there are no page reloads (Figure 4.1). Users can pan and zoom on the map, edit waypoint data, and edit their route. The "Add Waypoint" button drives updates to the page (weather data, route, etc.). Weather data is retrieved by asynchronous server calls that update page elements once they return.

4.3.2 Native Mobile Implementation

When developing the COLD-T native mobile application's user interface on the Android platform, we target small screen devices specifically ("phones" not "tablets"). We are unable to fit both the application input (map and form fields), and output (route table and clothing layer display) on the same screen. We split the application into two screens, one for input and one for output; we add tabs to the bottom of the page to allow the user to navigate between each screen. In order to maintain application familiarity, we preserve the names and functionality of

[3] *Simple* is a Java Object to XML mapping technology that works on both Android and Google App Engine. It could be used to create an XML to object translation layer for both environments to parse the NOAA web service response in to Java objects. The SOAP clients would still differ.

each form field and button. Where the program navigation differs from the original web application, we attempt to capitalize on the user's familiarity with the device by using native device navigation (tab panel and the Android back button).

4.3.3 User Agent Aware Web Based Implementation

When developing the COLD-T user agent aware web application, we target tablet style devices with slightly more screen real-estate. This allows us to maintain the exact same work-flow, buttons and functionality as the original web application. Once we identify a mobile user agent, we use conditional CSS to remove the header and footer and expand the UI elements to fit the entire screen. We also query the mobile device for its location and center the map on the device.

4.4 Conclusion

As we work through each application implementation, it becomes apparent that emerging mobile platforms are easily capable of running applications which access web services nearly as well as modern operating systems. Traditional enterprise services need to be changed little, if at all, in order for mobile applications to use them. The lowest common denominator for implementing an web service driven application capability across multiple mobile platforms is to develop a separate application for each. We find that there are few opportunities to share code across native platform implementations. However, we find that a user agent aware web application excels in this area. As we observed in our subsidiary research questions (Sections 4.2 and 4.3), a mobile web application can share code across multiple native implementations. The UI is tailored to the mobile client (size, positioning) but the server-side code and client-side logic remain the same. Equally, because it utilizes a single UI, the application look and feel is largely preserved across client devices.

The real benefit of this approach is realized because we offload all of the device dependence onto the mobile browser. Mobile browser developers ensure that HTML and JavaScript code executes the same on each device. This is not a new approach, in fact many early mobile application developers went this route only to find that their applications were unable to make use of the many APIs and sensors that make mobile devices and native mobile applications so rich and useful. The HTML 5 specification is changing this stalemate by offering access to native device functionality like geolocation, storage, and devices (camera, camcorder, microphone, etc.)

When an application does not need access to deep device APIs, HTML 5's device API coverage is good and getting better. We can see many of the building blocks of mobile application devel-

opment, outlined in Section 2, reflected in the HTML 5 specification. The framework described in section 2.2 [6] is basically a web browser. Many of the native capabilities that are accessible with the Rhomobile solution described in Section 2.2.1 are available with HTML 5. Finally, the ability to store data locally with a HTML5 compliant browser opens up the possibility of mobile aware web applications that can run locally without a live network connection.

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 5:
Conclusions and Future Work

In this chapter we summarize the results of our research questions detailed in Chapter 4 and present areas for future work on the COLD-T application. In Section 5.1 we discuss the different methods we explored to extend existing web based applications onto mobile platforms. In Section 5.2 we discuss planned future work to improve the COLD-T application to production-level quality.

5.1 Thesis Summary

This thesis focuses on methods for developing future web service based DoD applications which must be available as traditional desktop applications and on mobile devices. We explore different methods for extending application capability to mobile platforms with our case study application, COLD-T. The COLD-T application environment and requirements are such that it serves as a small scale representation of the general implementation challenges facing many DoD Programs of Record and IT acquisition commands. Specifically, finding effective ways for extending existing service based applications to mobile clients. While implementing the COLD-T application, we considered the client data access layer, the application UI design, and the TCO. We determined that native applications and mobile aware web applications are capable of consuming an existing web service without modifying the service. We explored the possibility of sharing a client-side data access module between application implementations and found it difficult and costly. Finally, we were able to create a mobile UI, on both mobile implementations, which maintained a look, feel, and work-flow that matched the original web application.

In general, we find no clear advantage between our mobile implementations when consuming existing web services. Both the native and mobile aware web application implementations of the COLD-T application were able to consume data from the NOAA SOAP web service using existing APIs and without modification to the web service. For the COLD-T application, because we can take advantage of the mobile browser's ability to provide geo-location and native storage, we find that a mobile aware web application is the most attractive option for quickly and cheaply extending the COLD-T capability to multiple mobile platforms in a cost-effective and efficient manner.

We found that, while developing a data access module, it is difficult to share code between an exiting web application and a native mobile application. Even when we specifically chose the web and native framework such that they shared a common programming language, we found that differences in compiling methods and deployment environments make it difficult to share anything more complicated than simple objects. As a result of this limited success and because there are few target environments that share a common language, we found that the benefit of sharing code between client data access modules decreases with each new target environment. We found that this problem is rendered moot when implementing a mobile aware web application as the client, and server side logic is shared across all target environments.

When developing the User Interface for COLD-T, we first developed a UI for the traditional web application from the user requirements and then attempted to duplicate the UI for both native implementations. The native Android device had the least amount of screen real estate. Our solution to this problem was to break the UI into two main screens while maintaining the same work-flow by conserving button actions and button names. For the mobile aware web application, we used CSS to expand the size of the UI when the application detected a mobile device. We removed all white space and resized the UI proportionally to fit the maximum screen real height and width. We were able to maintain exactly the same work-flow as the traditional web application. We found that we were able to maintain an easily recognizable User Interface on each mobile implementation.

We find that when extending an existing service based application to multiple, differing, mobile clients both native and mobile web application implementations are capable of consuming existing web services and maintaining a consistent UI. When choosing between the two implementation approaches, the most basic criteria is what native device APIs may need to be accessed in order to meet the application's requirements. If there is a requirement to access native device APIs that are not supported by emerging web technologies like HTML 5 then the native approach is most appropriate and native applications can be developed for each target mobile platform. However, it is undesirable to develop multiple completely different versions of the application, one version for each platform. Additionally, if we are considering extending *existing* application capability it's unlikely that there will be a requirement for a mobile specific sensor as most desktop computers don't have a camera, or GPS capability, etc. If this is the case, a mobile aware web application excels. Because a mobile aware web application maintains a single baseline, it's more cost effective in every aspect: development, testing, content management, deployment, and sustainment.

5.2 Future Work

FNMOC is continuing to develop new services to expose its gridded weather models and the Common Operational Research Environment (CORE) Lab is accepting COLD-T into its incubation area for Tactical Mission Planning research and development. Future work on the COLD-T application includes new data sources to improve pack-out recommendations and the implementation of client-side storage for improved offline operation.

5.2.1 Additional Data Sources

FNMOC has contracted the development of web services to expose its model data to a broader range of DoD entities. The COLD-T application will be a testbed for consuming these new services by highlighting their capability and providing feedback to the FNMOC developers. The planned data services are as follows.

1. FNMOC weather service: This service exposes the FNMOC model data in much the same way the NOAA weather data is exposed. The service returns textual values for requested weather parameters based on a geo-location and date-time. The FNMOC weather service expands the coverage of the COLD-T application from just the US Territories to the entire globe.
2. Climatology/Climatological (CLIMO) weather service: The CLIMO service exposes weather parameter predictions for a given geo-location and date-time based on historical averages. The CLIMO service functions as a backup data source for weather predictions when the planned mission date extends past the FNMOC weather forecast. CLIMO data can be requested as text per point or as a region file (for caching).
3. Elevation service: The elevation service exposes Digital Terrain Elevation Data (DTED) Level 0 elevation data for a given geo-location or region. Elevation data can be requested as text per point or as a region file (for caching).

5.2.2 Local Storage

One of the most exciting areas for future work on COLD-T is exploring the capabilities of the HTML5 storage APIs. We plan to find out how much of the COLD-T functionality can be maintained while in an off-line mode. There are three types of data that we would like to persist client-side: route data, map tiles, and gridded data.

1. Route data: All the information needed to reconstruct a planned route can be stored client-side in key-value pairs. Storing one or several routes client-side allows users to repeatedly

reference planned routes as the mission date gets closer and/or as weather predictions change. When a user returns to the application, it checks to see if local route data is present and loads the route(s) if it is.

2. Map tiles: Google Earth Portable Server and MapBox TileMill allow geographic areas of the globe to be "cut out," tiled, and stored locally. We will research how or if we can locally store map tiles for a region in order to enable off-line map interaction.
3. Gridded data: Similar to storing map tiles locally, we will explore ordering and caching geographic regions of elevation and CLIMO data in order to enable off-line predictions and route changes.

5.2.3 Occasionally Connected Applications

As more programs like COLD-T are extended to mobile platforms, architectures will undoubtedly be developed to allow the applications to continue to function in disconnected, intermittent and limited communication environments. This occasionally connected environment coupled with applications running on mobile devices is often termed "the tactical edge". A tactical edge environment can be described by three variables: device capability, network characteristics, and user preferences. As we move closer to the edge, device capability generally decreases, network characteristics become more constrained, and user preferences change. With these variables, we can measure a device's proximity to the tactical edge. We believe that a tactical edge rating can be extended beyond mobile devices. For example, by rating different DoD deployment environments, we may find alternative uses for occasionally connected applications. For example, the tactical edge profile of a Naval ship underway may be very similar to a mobile device, that is, low bandwidth, occasionally connected, etc. An application developed to function in a traditional mobile environment, capable of forward caching data, and functioning while disconnected could be just as applicable deployed on the watch floor of a Naval ship as on a hardened tablet in the field.

REFERENCES

[1] D. of Defense Chief Information Officer, "Information Management and Information Technology Stratigic Plan," *IEEE International Conference on E-Business Engineering*, p. 1, 2008-2009.

[2] L. G. D. Goldstein, *Combat Operator's Loadout Decision Tool (COLD-T)*. Monterey, CA: Fleet Numerical Meteorology and Oceanography Center (FNMOC), 2010.

[3] S. D. K. Hyun Jung La, "Balanced MVC Architecture for Developing Service-based Mobile Applications," *IEEE International Conference on E-Business Engineering*, pp. 292–299, 2010.

[4] B. L. Andre Charland, "Mobile Application Development: Web vs. Nativ." http://delivery.acm.org/10.1145/1970000/1968203/p20-charland.pdf, April 2011.

[5] A. S. Yuri Natchetoi, Viktor Kaufman, "Service-Oriented Architecture for Mobile Applications," *SAM*, pp. 27–32, May 2008.

[6] L. L. Biao Pan, Kun Xiao, "Component-based Mobile Web Application of Cross-platform," *IEEE International Conference on Computer and Information Technology*, vol. 10, pp. 2072–2077, 2010.

[7] N. M. Huaigu Wu, Louenas Hamdi, "TANGO: A Flexible Mobility-enabled Architecture for Online and Offline Mobile Enterprise applications," *International Conference on Mobile Data Management*, vol. 11, pp. 230–238, 2010.

[8] M. Milian, "U.S. Government, Military to get Secure Android Phones." http://edition.cnn.com/2012/02/03/tech/mobile/government-android-phones/index.html, Feb 2012.

THIS PAGE INTENTIONALLY LEFT BLANK

Initial Distribution List

1. Defense Technical Information Center
   Ft. Belvoir, Virginia
2. Dudly Knox Library
   Naval Postgraduate School
   Monterey, California
3. Marine Corps Representative
   Naval Postgraduate School
   Monterey, California
4. Directory, Training and Education, MCCDC, Code C46
   Quantico, Virginia
5. Marine Corps Tactical System Support Activity (Attn: Operations Officer)
   Camp Pendleton, California
   Officer students in the Operations Research Program are also required to show:
6. Director, Studies and Analysis Division, MCCDC, Code C45
   Quantico, Virginia
   Officer students in the Space Ops/Space Engineering Program or in the Information Warfare/Information Systems and Operations are also required to show:
7. Head, Information Operations and Space Integration Branch,
   PLI/PP&O/HQMC, Washington, DC